Lecture 8

Loops
Announcements for Today

Assignment 1

- We are still grading it
  - Will finish Sat at noon
- If you are close…
  - Will get feedback in CMS
  - Fix your assignment
- If you are very wrong…
  - You got an e-mail
  - Set up tutoring session
- **FINISH THE SURVEY**

Assignment 2

- Posted next **Friday**
  - We need one more lecture
  - Due after Fall Break
  - Do not start until A1 done

Reading

- Read Chapters 8, 10
- Focus on sections we skipped
Example: Summing the Elements of a List

```python
def sum(thelist):
    """Returns: the sum of all elements in thelist
Precondition: thelist is a list of all numbers (either floats or ints)""
    pass # Stub to be implemented
```

Remember our approach:
Outline first; then implement
Example: Summing the Elements of a List

```python
def sum(thelist):
    """Returns: the sum of all elements in thelist
    Precondition: thelist is a list of all numbers (either floats or ints)"
    
    # Create a variable to hold result (start at 0)
    # Add each list element to variable
    # Return the variable
```
Example: Summing the Elements of a List

```python
def sum(thelist):
    """Returns: the sum of all elements in thelist
    Precondition: thelist is a list of all numbers (either floats or ints)"
    result = 0
    result = result + thelist[0]
    result = result + thelist[1]
    ...
    return result
```

There is a problem here
Working with Sequences

• Sequences are potentially **unbounded**
  ▪ Number of elements inside them is not fixed
  ▪ Functions must handle sequences of different lengths
  ▪ **Example:** `sum([1,2,3])` vs. `sum([4,5,6,7,8,9,10])`

• Cannot process with **fixed** number of lines
  ▪ Each line of code can handle at most one element
  ▪ What if # of elements > # of lines of code?

• We need a new **control structure**
For Loops: Processing Sequences

# Print contents of seq
x = seq[0]
print(x)
x = seq[1]
print(x)
...
x = seq[len(seq)-1]
print(x)

• Remember:
  ▪ Cannot program ...

The for-loop:

```python
for x in seq:
    print(x)
```

• Key Concepts
  ▪ loop sequence: seq
  ▪ loop variable: x
  ▪ body: print(x)
  ▪ Also called repetend
For Loops: Processing Sequences

The for-loop:

```python
for x in seq:
    print(x)
```

- **loop sequence**: `seq`
- **loop variable**: `x`
- **body**: `print(x)`

To execute the for-loop:

1. Check if there is a “next” element of `loop sequence`
2. If not, terminate execution
3. Otherwise, put the element in the `loop variable`
4. Execute all of the `body`
5. Repeat as long as 1 is true
Example: Summing the Elements of a List

```python
def sum(thelist):
    """Returns: the sum of all elements in thelist
    Precondition: thelist is a list of all numbers (either floats or ints)"
    # Create a variable to hold result (start at 0)
    # Add each list element to variable
    # Return the variable
```
Example: Summing the Elements of a List

```python
def sum(thelist):
    """Returns: the sum of all elements in thelist
Precondition: thelist is a list of all numbers (either floats or ints)"
    result = 0
    for x in thelist:
        result = result + x
    return result
```

- **loop sequence:** thelist
- **loop variable:** x
- **body:** result = result + x
Example: Summing the Elements of a List

```python
def sum(thelist):
    """Returns: the sum of all elements in thelist
    Precondition: thelist is a list of all numbers (either floats or ints)""
    result = 0
    for x in thelist:
        result = result + x
    return result
```

Accumulator variable

- loop sequence: `thelist`
- loop variable: `x`
- body: `result = result + x`
def num_ints(thelist):
    # Returns: the number of ints in thelist
    # Precondition: thelist is a list of any mix of types
    # Create a variable to hold result (start at 0)
    # for each element in the list...
    # check if it is an int
    # add 1 if it is
    # Return the variable
def num_ints(thelist):
    """Returns: the number of ints in thelist
    Precondition: thelist is a list of any mix of types""
    result = 0
    for x in thelist:
        if type(x) == int:
            result = result + 1
    return result
Modifying the Contents of a List

```python
def add_one(thelist):
    """(Procedure) Adds 1 to every element in the list
    Precondition: thelist is a list of all numbers (either floats or ints)""

    for x in thelist:
        x = x + 1

    # procedure; no return
```

9/22/17 For Loops
For Loops and Call Frames

```python
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb."""
    for x in thelist:
        x = x + 1

add_one(seq):
```

```python
add_one
1
thelist
id4

seq
id4
0
5
1
4
2
7
```
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb."""
    for x in thelist:
        x = x + 1

add_one(seq):
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb."""
    for x in thelist:
        x = x + 1

add_one(seq):

Loop back to line 1

Increments x in frame
Does not affect folder
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb.""
    for x in thelist:
        x = x+1

add_one(seq):

seq id4
0 5
1 4
2 7

add_one
2
thelist id4
x 4

Next element stored in x.
Previous calculation lost.
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb.""
    for x in thelist:
        x = x + 1

add_one(seq):
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb."""
    for x in thelist:
        x = x + 1

add_one(seq):

seq

Next element stored in x.
Previous calculation lost.
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb."""
    for x in thelist:
        x = x + 1

add_one(seq):  

Loop back to line 1

```python
seq
  id4

0
  5
1
  4
2
  7

thelist
  id4
x
  8
```

9/22/17  For Loops
For Loops and Call Frames

```python
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb.""
    for x in thelist:
        x = x + 1

add_one(seq):
```

```
seq
  id4
  0 5
  1 4
  2 7

thelist
  id4
  x 8

Loop is completed.
Nothing new put in x.
```
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb.""
    for x in thelist:
        x = x+1

add_one(seq):

No changes to folder
def copy_add_one(thelist):
    """Returns: copy with 1 added to every element
    Precondition: thelist is a list of all numbers (either floats or ints)"
    mycopy = [] # accumulator
    for x in thelist:
        x = x+1
        mycopy.append(x) # add to end of accumulator
    return mycopy
How Can We Modify A List?

• **Never** modify loop var!
• This is an infinite loop:

```python
for x in thelist:
    thelist.append(1)
```

• Need a second sequence
• How about the *positions*?

```python
thelist = [5, 2, 7, 1]
theapos = [0, 1, 2, 3]
for x in theapos:
    thelist[x] = x+1
```

Try this in Python Tutor to see what happens
How Can We Modify A List?

- **Never** modify loop var!
- This is an infinite loop:
  
  ```python
  for x in thelist:
    thelist.append(1)
  ```

- Need a second sequence
- How about the *positions*?

```python
thelist = [5, 2, 7, 1]
thepos = [0, 1, 2, 3]

for x in thepos:
  thelist[x] = x+1
```

Try this in Python Tutor to see what happens
This is the Motivation for Iterators

- Iterators are objects
  - Contain data like a list
  - But cannot slice them
- Access data with `next()`
  - Function to get next value
  - Keeps going until end
  - Get an error if go too far
- Can convert back & forth
  - `myiter = iter(mylist)`
  - `mylist = list(myiter)`

Type/Class conversion

9/22/17 For Loops
Iterators and Lists

```python
>>> seq = [5, 4, 7]
>>> alt = iter(seq)
>>> next(alt)
5
>>> next(alt)
4
>>> next(alt)
7
>>> next(alt)
Traceback...
```
Iterators and For Loops

```python
>>> seq = [5, 4, 7]
>>> alt = iter(seq)
>>> for x in alt:
    print(x)
5
4
7
```

Just like looping over the list
Iterators and For Loops

```python
>>> seq = [5, 4, 7]
>>> alt = iter(seq)
>>> for x in alt:
    print(x)
5
4
7
```

But still not safe to modify iterator’s list

Just like looping over the list
The Range Iterator

- range(x)
  - Creates an iterator
  - Stores [0,1,…,x-1]
  - But not a list!
  - But try list(range(x))
- range(a,b)
  - Stores [a,…,b-1]
- range(a,b,n)
  - Stores [a,a+n,…,b-1]

- Very versatile tool
- Great for processing ints

```python
total = 0
# add the squares of ints # in range 2..200 to total
for x in range(2,201):
    total = total + x*x
```

Accumulator
def add_one(thelist):
    """(Procedure) Adds 1 to every element in the list
    Precondition: thelist is a list of all numbers
    (either floats or ints)""
    
    size = len(thelist)
    for k in range(size):
        thelist[k] = thelist[k]+1

    # procedure; no return
Important Concept in CS: Doing Things Repeatedly

1. Process each item in a sequence
   - Compute aggregate statistics for a dataset, such as the mean, median, standard deviation, etc.
   - Send everyone in a Facebook group an appointment time

2. Perform $n$ trials or get $n$ samples.
   - A4: draw a triangle six times to make a hexagon
   - Run a protein-folding simulation for $10^6$ time steps

3. Do something an unknown number of times
   - CUAUV team, vehicle keeps moving until reached its goal
Important Concept in CS: Doing Things Repeatedly

1. Process each item in a sequence
   - Compute aggregate statistics for a dataset, such as the mean, median, standard deviation, etc.
   - Send everyone in a Facebook group an appointment time

2. Perform $n$ trials or get $n$ samples.
   - **A4**: draw a triangle six times to make a hexagon
   - Run a protein-folding simulation for $10^6$ time steps

3. Do something an unknown number of times
   - CUAUV team, vehicle keeps moving until reached its goal

---

**For Loops**

```python
for x in sequence:
    process x
```

```python
for x in range(n):
    do next thing
```

**Cannot do this yet**

Impossible w/ Python for
Beyond Sequences: The while-loop

while <condition>:
    statement 1
    ...
    statement n

- Relationship to for-loop
  - Broader notion of “still stuff to do”
  - Must explicitly ensure condition becomes false
  - You explicitly manage what changes per iteration
print 'Before while'
count = 0
i = 0
while i < 3:
    print 'Start loop ' + str(i)
count = count + i
i = i + 1
print 'End loop'
print 'After while'

Output:

Before while
Start loop 0
End loop
Start loop 1
End loop
Start loop 2
End loop
After while
**while Versus for**

# process range b..c-1

```python
for k in range(b,c):
    process k

Must remember to increment
```

# process range b..c-1

```python
for k in range(b,c+1):
    process k
```

# process range b..c-1

```python
while k < c:
    process k
    k = k+1
```

```python
while k <= c:
    process k
    k = k+1
```
Cases to Use **while**

---

**Great for when you must modify the loop variable**

```python
# Remove all 3's from list t
i = 0
while i < len(t):
    # no 3's in t[0..i-1]
    if t[i] == 3:
        del t[i]
    else:
        i = i+1

# Remove all 3's from list t
while 3 in t:
    t.remove(3)
```

9/22/17

For Loops
Cases to Use **while**

Great for when you must **modify** the loop variable

```python
# Remove all 3's from list t
i = 0

while i < len(t):
    # no 3's in t[0..i-1]
    if t[i] == 3:
        del t[i]
    else:
        i += 1
```

```python
# Remove all 3's from list t
while 3 in t:
    t.remove(3)
```

The stopping condition is not a numerical counter this time. Simplifies code a lot.

Stopping point keeps changing.

9/22/17

For Loops
Cases to Use while

• Want square root of $c$
  ▪ Make poly $f(x) = x^2 - c$
  ▪ Want root of the poly
    ($x$ such that $f(x)$ is 0)

• Use Newton’s Method
  ▪ $x_0 = \text{GUESS } (c/2??)$
  ▪ $x_{n+1} = x_n - f(x_n)/f'(x_n)$
    $= x_n - (x_n x_n - c)/(2x_n)$
    $= x_n - x_n/2 + c/2x_n$
    $= x_n/2 + c/2x_n$
  ▪ Stop when $x_n$ good enough

```python
def sqrt(c):
    """Return: square root of $c$
    Uses Newton’s method
    Pre: $c \geq 0$ (int or float)"
    x = c/2
    # Check for convergence
    while abs(x*x - c) > 1e-6:
        # Get $x_{n+1}$ from $x_n$
        x = x / 2 + c / (2*x)
    return x
```

9/22/17
For Loops